FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ústav telekomunikací

Multimediální systémy
Cvičení č. 2

Garant kurzu: **doc. Ing. Petr Číka PhD.**

Cvičící: **Ing. Milan Bubniak**
Ing. David Kohout

Akademický rok: 2022/2023

Osnova

- Opakování základů jazyka Java
- Samostatná práce na základy Javy

Hierarchie zdrojového kódu

- Zdrojové kódy tříd rozděleny do tzv. **balíčků (packages)**
- Prevence konfliktů v názvech tříd
- Řízení přístupu k třídám
  - Default/protected

```
// import tridy Vector z balicku util
import java.util.vector;

// import všech trid z balicku util
import java.util.*;
```

Primitivní datové typy

- Java = staticky typovaný jazyk
- 8 primitivních datových typů
- Základní nositelé informace

Typ	Minimum	Maximum	Velikost
byte	-128	127	1 B
short	-32 768	32 767	2 B
int	-2 147 483 648	2 147 483 647	4 B
long	-9 223 372 036 854 775 808	9 223 372 036 854 775 807	8 B

Typ	Minimum	Maximum	Velikost
float	*plovoucí*	*plovoucí*	4 B
double	*plovoucí*	*plovoucí*	8 B
boolean	false	true	1 b
char	0	65 535	2 B

Referenční datové typy

- **String**, **Array**, **Class**, **Interface**, **Enumeration**…
  - (vše mimo primitivní typy)
- Dědí z „pratřídy“ **java.lang.Object**
- Jedná se o objekty
- Disponují **metodami**
- Proměnná je defacto pouze ukazatelem (referencí) na místo v paměti, ve kterém se nachází objekt daného typu

Operátory

- Aritmetické
  `+` `-` `*` `/` `%` `++` `--`
- Přiřazovací
  `=` `+=` `-=` `*=` `/=` `%=` `&=` `|=` `^=` `>>=` `<<=`
- Porovnávací
  `==` `!=` `>` `<` `>=` `<=`
- Logické
  `&&` `||` `!`
- Bitové
  `&` `^` `|` `<<` `>>` `~`
- Ternární
  `?:`

Řídící struktury

Podmínky

- if
- if-else
- if-else-if

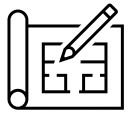
Cykly

- for
- foreach
- while
- do-while

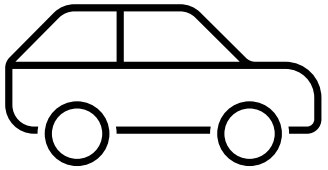
Návraty

- return
- break
- continue

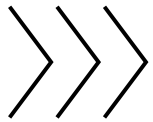
Třída vs. objekt



Třída



Auto



Objekt
(instance třídy)

Škoda Fabia

Volkswagen Passat

Volvo V60

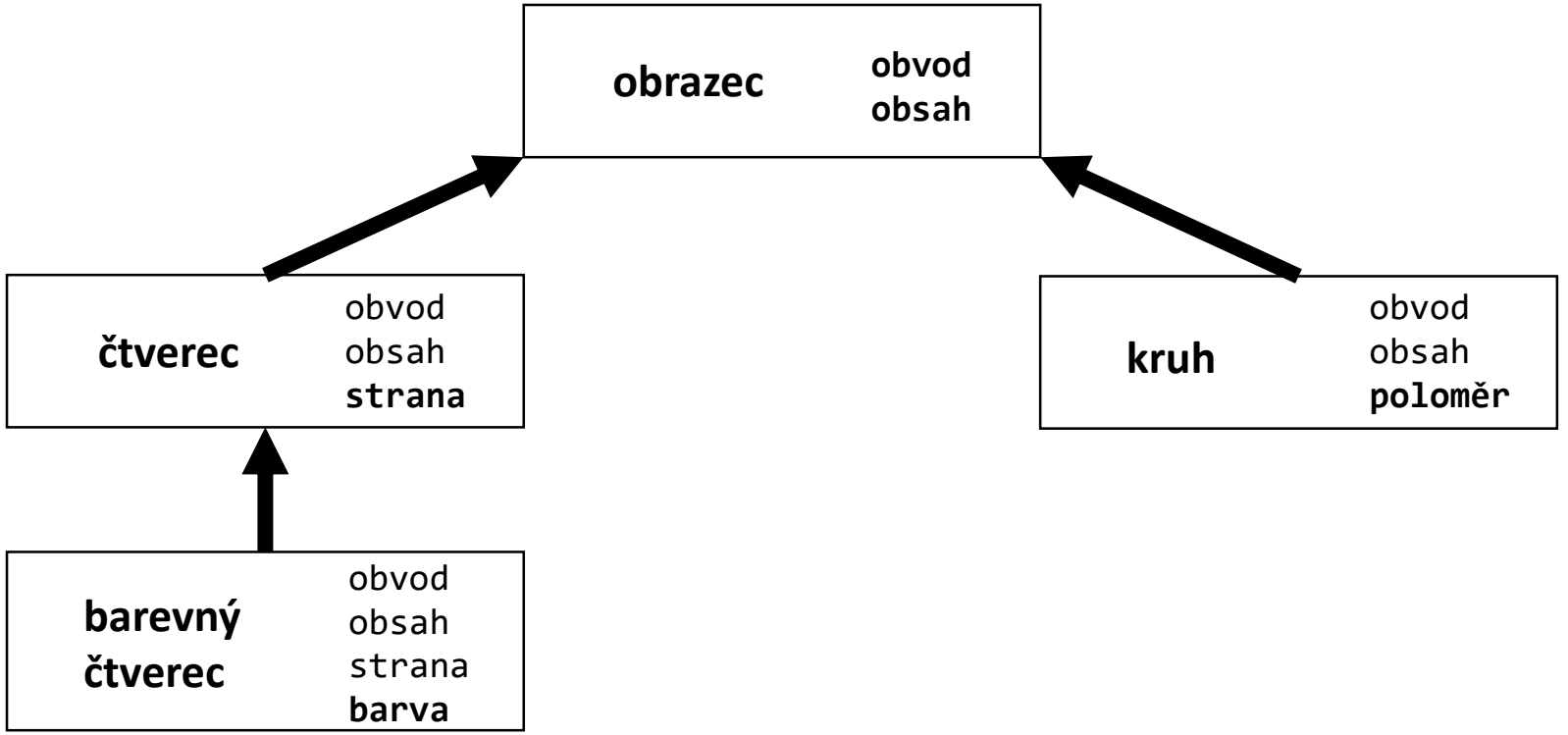
Metoda vs. funkce

- Java = objektový jazyk, nejsou v ní tedy funkce
- Metody se **vždy** nacházejí v instancované či nadřazené třídě

- Funkce jsou samostatně – **nejsou** v žádné nadřazené třídě
- Např. jazyk Python = lze programovat funkcionálně i objektově

Dědičnost

- Klíčové slovo **extends**
- Lze přímo dědit pouze z jedné nadřazené třídy (na rozdíl od např. C++)



Rozhraní

- Klíčové slovo **implements**
- Rozhraní (**interface**) definuje metody, které musí obsahovat implementující třída
- Rozhraní může obsahovat i definice konstant
- Třída může implementovat **libovolný počet** rozhraní

```
// Ukazka jednoducheho rozhrani
interface MyInterface {

    final int a = 10;          // deklarace konstanty

    void display();            // deklarace abstraktni metody
}
```

Konstruktory

- Speciální metoda volaná **při vytváření objektu**
- Má stejný název, jako daná třída
- Lze mít jeden či více konstruktorů
  - Rozdílné chování dle typu a počtu parametrů
  - Volání jiných konstruktorů – klíčové slovo **this**

```
public class Testovaci {
  int x;

  // konstruktor tridy s jednim parametrem
  public Testovaci(int y) {
    x = y;
  }
}
```

Statické proměnné a metody

- Lze definovat statické proměnné a metody
- Klíčové slovo **static**
- Statická proměnná
  - Informace „sdílená“ mezi **všemi instancemi** dané třídy
  - V paměti uložena pouze jednou
  - Změna proměnné z jedné instance se projeví ve všech ostatních instancích
- Statická metoda
  - Může pracovat pouze se svými parametry a se statickými proměnnými

Modifikátory přístupu

- Ovlivňují viditelnost tříd, proměnných, konstruktorů a metod
- **default** (neuvedeno nic)**, private, protected, public**

	default	private	protected	public
Třída	✓	✓	✓	✓
Podtřída v balíčku	✓	✗	✓	✓
Jiná třída v balíčku	✓	✗	✓	✓
Podtřída mimo balíček	✗	✗	✓	✓
Jiná třída mimo balíček	✗	✗	✗	✓

Anotace

- Anotace slouží jako zápis přiřazující nějakému elementu doplňující informaci mimo běžný kód
- Příklady klasických anotací:
  - @Override
    - Přepsání celé metody nebo její rozšíření v podtřídě
    - Rozšíření klíčovým slovem **super**
  - @Deprecated
    - Označení zastaralých metod
  - @SuppressWarnings
    - Potlačení varování

Kolekce

- Pole (**Array**)
  - Pevná délka
  - Prvky v paměti za sebou
- Seznamy (**List**)
  - Proměnná délka
  - Lze vkládat a odebírat z jakékoli části seznamu
- Asociativní pole (**Map**)
  - Sestávající z dvojic klíč-hodnota
  - Rychlý přístup k hodnotě pomocí klíče

Výjimky

- Ošetření chyb, které se mohou objevit za běhu programu
- Na místě výskytu chyby:
  1. Dojde k pozastavení zpracování programu
  2. Vytvoří se objekt výjimky dědící ze třídy **Throwable**, resp. **Exception** (vzácně **Error**)
     - Objekt obsahuje **podrobné informace o chybě**
  3. Výjimka je zachycena ošetřujícím kódem, tento kód je vykonán
  4. Program pokračuje v místě kódu po ošetření výjimky
- Vyvolání výjimky klíčovým slovem **throw**
- Zachycení výjimky blokem **try-catch**
  - **try** – „nebezpečný“ úsek kódu
  - **catch** – ošetření výjimky

Prostor pro dotazy

Společné opakování

- Viz kód na plátně

Samostatný úkol

- Vytvořte novou třídu s názvem **MapClass**:
  - Třída bude obsahovat **privátní** proměnnou typu **asociativní pole.**
    - (Např. typ **HashMap<Integer, String>**, konkrétní implementaci můžete zvolit jakoukoli).
  - Tuto proměnnou inicializujte tak, aby **obsahovala prázdné asociativní pole**. Inicializaci proveďte **při vytváření instance** třídy MapClass.
  - S asociativním polem se bude pracovat pomocí následujících metod:
    - Metoda **public void store(Integer id, String value)** vloží do pole nový prvek (nové mapování mezi klíčem specifikovaný parametrem *id* a hodnotou specifikovanou parametrem *value*). Pokud je prvek s daným ID **již v poli namapován**, vraťte výjimku **ArrayStoreException.**
    - Metoda **public String getValue(Integer id)** vrátí hodnotu pro klíč specifikovaný parametrem *id*. Pokud prvek s daným ID **v poli není namapován**, vraťte výjimku **NoSuchFieldException**.
    - Metoda **public void deleteKey(Integer id)** smaže mapování pro klíč specifikovaný parametrem *id* a smaže prvek z pole. Pokud prvek s daným ID **v poli není namapován**, vraťte výjimku **NoSuchFieldException**.
    - Metoda **public int getSize()** vrátí aktuální počet prvků v poli.
    - Metoda **public void print()** vytiskne na standardní výstup (*System.out*) obsah asociativního pole tak, že každá dvojice klíč-hodnota bude na samostatném řádku, a řádky budou formátovány jako „**<klíč>-><hodnota>**“. Příklad:
      - Do pole vložíme mapování – klíč: “100”, hodnota: “stovecka”,
      - Zavoláme-li metodu print, vytiskne se na výstup “100->stovecka”.

Dokumentace HashMap<K, V>: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/HashMap.html